Agent JS – Ügynökvezérelt programozás JavaScripttel

Visnovitz Márton[1], Horváth Győző[2]

{[1]visnovitz.marton,[2]horvath.gyozo}@inf.elte.hu
ELTE IK

Absztrakt. A Logo programozási nyelv nagy hagyományokkal rendelkezik az informatika oktatásában Magyarországon, diákok generációit vezette be a programozás világába. A teknőcgrafika, illetve a grafikus programozás a mai napig nagyon népszerű módszerek a programozás alapjainak bemutatására. Idővel számos új eszköz, nyelv, környezet jelent meg, mely lehetővé tette az ilyen jellegű programok készítését (pl. Scratch, Python with Turtle), sőt a Logo nyelv egyik változatával, a NetLogo-val már nemcsak egy, hanem több teknőcből (ügynökből) álló rendszereket, komplex modelleket is készíthetünk. Ebben a cikkben egy fejlesztés alatt álló lehetőséget mutatunk be, mely lehetővé teszi, hogy JavaScript programozási nyelven, könnyedén készítsünk ilyen, akár több ügynökös programokat, modelleket. Ezáltal lehetővé válik, hogy a Python with Turtle rendszerhez hasonlóan, egy az iparban is jelentős, de az oktatásban is egyre inkább elterjedt nyelven tudjuk teknőcgrafikán, modellezésen keresztül programozni tanítani a diákokat.

Kulcsszavak: teknőcgrafika, ügynökvezérelt modellezés, JavaScript, Logo, NetLogo

Bevezetés, motiváció

A Logo programozási nyelv[1] 1967-es megjelenése óta hatalmas hatással volt a programozás oktatására. A nyelv, és a hozzá tartozó programozási környezetek egyszerű eszközökkel teszik lehetővé komplex grafikák készítését, a programozás számos eszközének, fogalmának megismerését [1, 2]. A teknőcgrafika mint témakör direkt módon volt jelen az előző, 2012-es Nemzeti Alaptantervben (NAT) [3], az új, 2020-as NAT-ban [4] a robotika témakör részeként jelenik meg. Magyarországon a mai napig népszerű módszer, rengeteg tanár alkalmazza az iskolában. A legutóbbi, 2022. évi Országos Grafikus Programozási Versenyen is közel 1000 diák indult el[2].

A Logo programozási nyelv és a teknőcgrafika alapja egy szereplő (teknőc, ügynök) számítógépes programmal való vezérlése, irányítása. Ennek a megközelítésnek az egyik lehetséges továbbfejlesztési iránya az úgynevezett többügynökös (többteknőcös) modellek programozása. Ezekben a programokban már nem csak egy, hanem több ügynököt vezérelhetünk, minden ügynök egy önálló szereplő a modellben. Az ilyen jellegű programok létrehozására is léteznek környezetek, az ügynökvezérelt modellezés oktatási felhasználásának egyik úttörője a NetLogo[3]. Megalkotója, Uri Wilensky, a Logo nyelv szellemiségében egy olyan környezetet alkotott, melyben „alacsony a belépési küszöb, de a határ a csillagos ég" (eredeti: "low threshold and no ceiling"), és ami remekül használható a programozás tanítására, a számítógépes gondolkodás fejlesztésére [5, 6, 7]. Az ügynökvezérelt modellek készítésének egyik nagy előnye, hogy segítségükkel nemcsak programozást taníthatunk, hanem változatos modellek segítségével kombinálhatjuk az informatikai ismeretek tanítását más tantárgyakkal is. A

[1] https://el.media.mit.edu/logo-foundation/what_is_logo/logo_programming.html

[2] https://njszt.hu/hu/news/2022-02-08/1000-gyerek-indult-el-grafikus-programozas-versenyen

[3] https://ccl.northwestern.edu/netlogo/

NetLogo környezethez készített modell-könyvtárban[4] találhatunk példákat többek között biológia, kémia, fizika, matematika és társadalomtudományok témakörökben is.

A teknőcgrafika, illetve a hozzá kapcsolódó robotika, modellezés/szimuláció, fontos szerepet töltenek be a programozás oktatásában. Ezen stratégiák lehetőséget biztosítanak, hogy a tanított korosztálynak megfelelő módszerekkel tudjuk tanítani az informatikával, számítástudománnyal kapcsolatos ismereteket [8]. Ezeken a területeken a Logo megjelenése óta rengeteg új lehetőség, eszköz, környezet látta meg a napvilágot, mint például a korábban említett NetLogo, a Scratch[5], mely blokkprogramozással tette lehetővé a szereplők irányítását, vagy a Python with Turtle[6] könyvtár, mely a népszerű Python programozási nyelven biztosít eszközöket ahhoz, hogy teknőcgrafikus programokat hozzunk létre. Egy érdekes új kezdeményezés továbbá az XLogo[7] webes környezet, mely különböző korcsoportoknak biztosít egyre bővülő eszköztárat teknőcgrafikus programok létrehozására: a legkisebb korosztályoknak blokkalapú programozással, az idősebbeknek pedig a Python with Turtle környezet segítségével.

Az új eszközöket illetően megfigyelhető trend a programozásoktatás világában a blokkalapú programozás, valamint a szkriptnyelvek használata. A Python mellett a másik szkriptnyelv, amivel gyakran találkozhatunk oktatási környezetekben a JavaScript (pl. CodeCombat[8], Grasshopper[9], Khan Academy[10]). Ez a szövegalapú programozási nyelv követi a blokk alapú kódolást a Micro:bit[11] esetében is. Több tanulmány is igazolja, hogy a JavaScript jó tulajdonságokkal rendelkezik a programozás tanításához [9, 10].

A Python mint (az oktatásban is) népszerű szkriptnyelv már rendelkezik teknőcgrafikai lehetőséggel, mellette a JavaScript nyelv egy új irányt jelenthet a programozott grafikában, ügynökvezérlet modellezésben. Ennek támogatására jött létre az Agent JS projekt.

Célok, filozófia

Az Agent JS projekt célja egy olyan JavaScript programkönyvtár készítése, ami lehetővé teszi ügynökvezérelt programok készítését JavaScript nyelven, különböző komplexitási szinteken. Segítségével könnyedén készíthetünk ábrákat teknőcgrafika segítségével a Logo nyelvhez hasonlóan, vagy többügynökös, illetve sejtautomata modelleket a NetLogo mintájára. Ehhez a könyvtárban hozzáférhető osztályokat kell kombinálnunk a JavaScript nyelv natív eszközeivel (pl. ciklusok, elágazások).

A könyvtár által biztosított eszköztár motivációját a Python with Turtle és a NetLogo környezetek adták. Célunk az, hogy az ezek által nyújtott lehetőségeket minél jobban lefedjük. Egy olyan könyvtárat szeretnénk létrehozni, ami sokféle lehetőséget kínál, de azok használatára nem kényszerít rá. Ennek megfelelően a könyvtár úgy került kialakításra, hogy lehetőség van csak a számunkra éppen szükséges elemek betöltésére. Így például, ha egy egyszerű, „egyteknőcös” rajzot szeretnénk készíteni, akkor nincs szükségünk az időzítő osztály betöltésére.

---

[4] https://ccl.northwestern.edu/netlogo/models/index.cgi

[5] https://scratch.mit.edu/

[6] https://docs.python.org/3/library/turtle.html

[7] https://xlogo.inf.ethz.ch/

[8] https://codecombat.com/

[9] https://grasshopper.app/

[10] https://www.khanacademy.org/

[11] https://microbit.org/

A könyvtár számos olyan kisegítő lehetőséget is tartalmaz, amelyek megkönnyítik bizonyos programozási feladatok megoldását, de ezek használata is opcionális a tanítási céltól függően. Például, a könyvtár tartalmaz segédfüggvényeket a szögek *fok* és *radián* közötti átváltásához. Ezeket szabadon használhatjuk egy alacsonyabb évfolyamon, ahol a diákok még nem feltétlenül tanulták az erre vonatkozó matematikai ismereteket, de gimnáziumban a tanulók már maguk is implementálni tudják ezeket a függvényeket a JavaScript nyelv beépített eszközeivel.

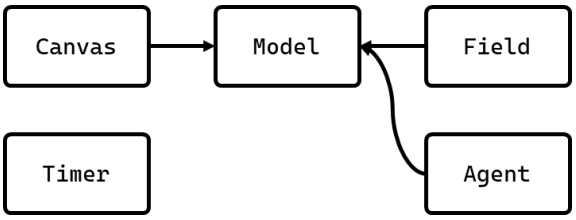
Felépítés, technológia

A JavaScript programozási nyelv jó alapokat ad grafikus programkönyvtárak készítéséhez. A böngészőben elérhető Canvas API[12] révén rendelkezésre áll egy beépített eszköztár a programozott rajzolásra, segítségével rendkívül komplex grafikák is készíthetők natív eszközökkel. Használatában az egyetlen dolog, ami nehézséget jelent kezdők számára a HTML nyelv és az azt vezérlő JavaScript API közötti összefüggés megértése. A fejlesztés során igyekeztünk minél közelebb maradni a natív eszköztárhoz, de kiküszöbölni az azzal kapcsolatos nehézségeket. Ennek megfelelően az Agent JS könyvtár több vékonyabb és vastagabb absztrakciós réteget épít a böngésző natív programozási interfészei fölé.

A könyvtár a modern webes sztenderdeknek megfelelően JavaScript modulként[13] lett megvalósítva, és a GitHub[14] kódtár segítségével van publikálva. Betöltése URL-en keresztül lehetséges, de amennyiben letöltjük a forráskódot a helyi számítógépre, internethozzáférés nélkül is használható. Az egyes osztályok és segédfüggvények úgynevezett „named import”-ok[15] formájában férhetőek hozzá.

```
import { Model } from "https://vimtaai.github.io/agent/lib/index.js";
```

A könyvtár alapját a `Canvas` osztály [11] adja, mely a natív `HTMLCanvasElement` típuson (`<canvas>` HTML tag) alapszik, annak lehetőségeit kombinálja a Canvas API programozó interfészével, a `RenderingContext2D` osztállyal. Erre, és az időzítők használatát megkönnyítő `Timer` osztály [11] fölé épül a `Model` absztrakció. Ez az osztály reprezentálja az ügynökvezérelt modellünket. A `Model` osztály példányának létrehozásakor automatikusan létrejön a modell megjelenéséért felelős `Canvas` példány, illetve automatikusan feltöltődik a modell a sejtautomata-elvű szimulációk működéséhez szükséges `Field` objektumokkal is. Ezek a `Field` objektumok a NetLogo környezet „patch” típusú ügynökeinek az analógiái. A modellhez ezen kívül van lehetőségünk ügynököket („teknőcöket”) hozzáadni az `Agent` osztály példányainak létrehozásával. A könyvtárban definiált osztályok kapcsolatát az 1. ábra szemlélteti.



1. ábra: Az Agent JS könyvtár osztályai

[12] https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

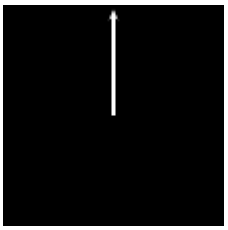
[13] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules

[14] https://github.com/

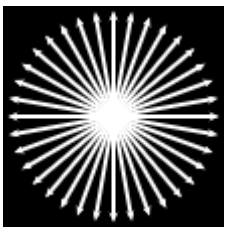
[15] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import#named_import

A hozzáadott ügynökök számától függően készíthetünk sejtautomata-elvű modelleket (nincs mozgó ügynök, csak egyhelyben álló ügynökök – mezők – vannak), egyszerű teknőcgrafikát (1 mozgó ügynök) vagy többügynökös modellt (tetszőleges számú mozgó ügynök) . Ezek a lehetőségek tetszés szerint kombinálhatóak is. A könyvtár használatára példát a 2. ábra mutat.

```
const model = new Model();
const agent = new Agent({
  x: model.centerX,
  y: model.centerY
});

model.addAgent(agent);
agent.putPenDown();
agent.forward(10);
```



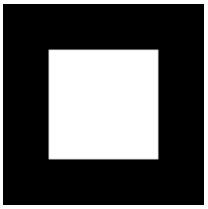
```
const model = new Model();

for (let i = 0; i < 36; i++) {
  const agent = new Agent({
    x: model.centerX,
    y: model.centerY,
    heading: i * 10
  });

  model.addAgent(agent);
  agent.putPenDown();
  agent.forward(10);
}
```



2. ábra: Példa együgynökös és sokügynökös rajzolásra

Habár a könyvtár fejlesztésének elsődleges fókusza a teknőcgrafika és az ügynökvezérelt programozás volt, a komponensei úgy lettek felépítve, hogy az alacsonyabb szintű absztrakciók külön is használhatóak legyenek. Ezek segítségével – teknőcök és a modellek nélkül – kizárólag a rajzolást és az animációt támogató osztályok (`Canvas` és `Timer`), valamint natív JavaScript függvények segítségével is van lehetőség ábrák, animációk vagy akár játékok készítésére (3. ábra) [12]. Ezek az alacsonyabb szintű absztrakciók korábbi tapasztalataink alapján úgy lettek kialakítva, hogy a natív JavaScript eszközöket csupán annyira „okosítsák fel”, hogy csak a használat szempontjából kényelmetlen, a megértést nehezítő részek legyenek elfedve, de amennyire lehetséges, natív függvényhívásokkal legyenek programozhatóak [13].

```
const canvas = new Canvas();

canvas.fillStyle = "black";
canvas.fillRect(0, 0, 100, 100);
canvas.fillStyle = "white";
canvas.fillRect(25, 25, 50, 50);
```



3. ábra: Egyszerű alakzatok rajzolása az "ügynök" absztrakció nélkül, natív eszközökkel

A könyvtár az ügynökvezérelt programozáshoz szükséges absztrakciókon túl számos segédfüggvényt tartalmaz, melyek megkönnyíthetik a programjaink elkészítését. Ezek a függvények három kategóriába vannak sorolva: véletlenszám-generáló függvények, geometriai függvények, illetve típushelyesség-ellenőrző függvények. Ezek használata teljesen opcionális, működésüket a diákok maguk is leprogramozhatják, de a kezdőbbek, fiatalabbak számára segítséget nyújthatnak abban, hogy könnyebben készíthessenek minél látványosabb programokat.

A `Canvas` és a `Model` absztrakciókhoz kihasználtuk a JavaScript nyelv és a böngészők egy viszonylag új, és elterjedőben lévő lehetőségét, a webkomponensek[16] definiálását. Ezzel a módszerrel saját HTML elemeket tudunk létrehozni, vagy egy meglévő elem lehetőségeit bővíteni. Ilyen formán a `Canvas` típus a natív `<canvas>` elem kibővítésén, a `Model` osztály pedig egy egyedileg definiált HTML elemen alapul.

Habár az Agent JS könyvtár által biztosított osztályok használatához HTML ismeretekre nincs szükség, a webes, böngészőbeli környezet jellegéből adódóan mégis szükség van egy minimális HTML fájlra programjaink futtatásához. Ahhoz, hogy a böngészőben JavaScript kódot futtassunk, a forráskódunkat tartalmazó `.js` kiterjesztésű fájlt vagy külső szkripfájlként kell hivatkozni, vagy közvetlenül a HTML fájlba kell írni a forráskódot `<script>` tag-ek közé. Ezen túl további HTML elemek hozzáadására nincs szükség, azok automatikusan beillesztésre kerülnek az oldalra amikor szükséges.

```
<script type="module">
// Ide írjuk a forráskódot
</script>
```

```
<!-- A program.js fájlba írjuk a forráskódot -->
<script type="module" src="program.js"></script>
```

Természetesen ettől függetlenül az oldalhoz van lehetőség további HTML elemeket beilleszteni amennyiben szeretnénk. Ezeket bemenetként vagy kimenetként használhatjuk a programunkban, hasonlóan a NetLogo környezet vezérlőihez. A programunk ilyen irányú kibővítése – azon túl, hogy a modellünk tudását bővíti – kiváló lehetőség a HTML nyelv, illetve a natív JavaScript programozás további részeinek megismerésére. Egyszerű eseménykezeléssel lehetőségünk nyílik a modellünket vezérlő gombokat, a modell paramétereit tartalmazó beviteli mezőket, vagy a modellünk állapotát kijelző mezőket létrehozni.

Dokumentáció

Az Agent JS mindenki számára szabadon hozzáférhető, nyílt forráskódú szoftver. A forráskód[17] és a dokumentáció[18] bárki számára elérhető a GitHub online kódtárban, bővítésükhöz, fejlesztésükhöz bárki hozzájárulhat.

A dokumentáció tartalmazza az osztályok és segédfüggvények részletes leírását, illetve rövid összegzést a könyvtár használatáról. Ezen túl a dokumentáció részét képezi néhány valós, élő példa,

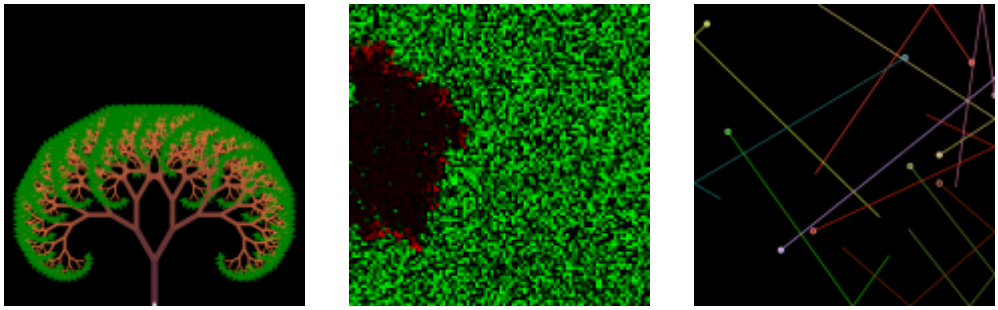
[16] https://developer.mozilla.org/en-US/docs/Web/Web_Components

[17] https://github.com/vimtaai/agent

[18] https://vimtaai.github.io/agent/

melyek a könyvtárban rejlő lehetőségeket mutatják be (4. ábra). Az alábbi feladattípusok mindegyikéhez készültek példák[19]:

- mozaikrajzolás, rekurzív ábra rajzolása teknőccel (parketta minta, rekurzív fa)
- sejtautomata modell (élet játéka, erdőtűzmodell)
- többügynökös modell (labdák pattogása, naprendszermodell)



4. ábra: Különféle rajzok és modellek az Agent JS könyvtárral
(balról jobbra: rekurzív fa, erdőtűzmodell, labdák pattogása)

Ezeken a példákon keresztül láthatjuk, hogy hogyan oldhatók meg különböző jellegű és komplexitású feladatok az Agent JS segítségével. A feladatok megoldásra révén szerzett tapasztaltokat felhasználtuk a könyvtár továbbfejlesztésére.

Jövőbeli tervek

Habár az Agent JS könyvtár már jelenlegi formájában is alkalmas számos programozásoktatási stratégia alkalmazására, a jövőben további lehetőségekkel tervezzük fejleszteni. A fejlesztések elsődleges iránya, hogy a könyvtár alkalmas legyen a Python with Turtle és a NetLogo környezetek által nyújtott lehetőségeket minél jobban lefedni. Reményeink szerint, ha a könyvtárat elkezdik használni az iskolákban, akkor a tanárok tapasztalatai alapján további ötleteket kaphatunk arra vonatkozóan, hogy mivel lehetne a könyvtárat használhatóbbá, kényelmesebbé tenni.

Ahhoz, hogy minél kevesebb előkészülettel legyen lehetőség elkezdeni dolgozni a környezettel, a terveink között szerepel az is, hogy az Agent JS könyvtárat legyen lehetőség valamilyen online szerkesztőprogram segítségével, közvetlenül a böngészőből használni. Ezáltal külső fejlesztőkörnyezet és a HTML fájlok használatára nélkül is lehetne modelleket készíteni.

Amennyiben a könyvtárnak sikerül nagyobb népszerűségre szert tenni, szeretnénk kezdeményezni, hogy az Országos Grafikus Programozási Verseny választható környezetei között szerepeljen az Agent JS is.

Összefoglalás

A teknőcgrafika a mai napig népszerű módja a programozásoktatásnak, használatát több programozási nyelv és környezet is támogatja. Az Agent JS projekt azért jött létre, hogy JavaScript nyelven is legyen lehetőségünk teknőcgrafikus alkalmazásokat, ügynökvezérlet modelleket készíteni. Hasz-

[19] https://vimtaai.github.io/agent/examples

nálatával a tanárok egy újabb eszközt kapnak ahhoz, hogy a tanulói igényekre szabott módszerekkel tanítsanak programozást, valamint programozási témaköröket, ismeretköröket kapcsoljanak össze.

Irodalom

1. Seymour Papert, Cynthia Solomon: *Twenty Things to Do with a Computer*, Artificial Intelligence Memo No. 248, (1971)
2. Seymour Papert: *Mindstorms. Children, Computers and Powerful Ideas*, Basic Books Inc., Harper Colophon Books, (1981)
3. *Nemzeti Alaptanterv 2012. (110/2012. (VI. 4.) Kormányrendelet)*, Magyar Közlöny 2012 66. sz. (2012) https://magyarkozlony.hu/dokumentumok/f8260c6149a4ab7ff14dea4fd427f10a7dc972f8/letoltes
4. *Nemzeti Alaptanterv 2020. (5/2020.(I. 31.) Kormányrendelet)*, Magyar Közlöny 2020 17. sz. (2020), https://magyarkozlony.hu/dokumentumok/3288b6548a740b9c8daf918a399a0bed1985db0f/letoltes
5. Seth Tisue, Uri Wilensky: *NetLogo: A Simple Environment for Modeling Complexity*. Proceedings of International conference on complex systems, Boston (2004) pp. 16-21.
6. Uri Wilensky: *Modeling nature's emergent patterns with multi-agent languages*. Proceedings of the Eurologo 2001 Conference, Linz (2001).
7. Bernát Péter: *Modelling and simulation in education and the NetLogo simulation environment*, Teaching Mathematics and Computer Science, 2014. Vol. 12, Num. 2, pp. 229-240.
8. Bernát Péter, Zsakó László: *Methods of teaching programming – strategy*, XXX. DIDMATTECH (2017), Trnava. pp. 40-50.
9. Visnovitz Márton: *Szöveges programozási nyelvek a közoktatásban*, Szakdolgozat
10. Horváth, Gy., Menyhárt, L. (2014). *Teaching introductory programming with JavaScript in higher education*, Proceedings of the 9th International Conference on Applied Informatics, Eger, Hungary. pp. 339-350.
11. Visnovitz Márton, Horváth Győző: *JavaScript könyvtárak programozott rajzolás alapú tanulás támogatásához,* InfoDidact 2020, (2020)
12. Horváth Győző, Menyhárt László, Zsakó László: *Viewpoints of programming didactics at a web game implementation,* XXIX. DIDMATTECH (2016) Budapest, pp. 79–88
13. Visnovitz Márton, Horváth Győző., *A Constructionist Approach to Learn Coding with Programming Canvases in the Web Browser*, *CONSTRUCTIONISM 2020*, (2020), pp. 1–8